|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ДИСЦИПЛИНА «Компьютерная графика»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа №4 по теме:**“РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ АЛГОРИТМОВ ГЕНЕРАЦИИ ОКРУЖНОСТИ И ЭЛЛИПСА”

|  |  |
| --- | --- |
| Студент: Свиридов И.Р.  Группа: ИУ7-45Б  Оценка (баллы) \_\_\_\_\_\_\_  Преподаватель: Куров А. В. |  |

Москва.

2020 г.

**Цель работы:**

Реализация алгоритмов построения окружности, исследование и сравнение визуальных и временных характеристик алгоритмов.

**Техническое задание:**

1.Реализовать алгоритмы построения окружности на основе

- Канонического уравнения X^2+Y^2=R^2

- Параметрического уравнения X=Rcost, Y=Rsint

- Алгоритма Брезенхема

- Алгоритма средней точки

- построение окружности с помощью библиотечной функции

Пользователь выбирает из списка определенный алгоритм, задает координаты центра, радиус, цвет рисования.

Визуальные характеристики исследуются путем рисования той же окружности цветом фона, но с помощью другого алгоритма.

2. Реализовать алгоритмы построения эллипса на основе

- Канонического уравнения X^2/a^2+Y^2/b^2=1

- Параметрического уравнения X=acost, Y=bsint

- Алгоритма Брезенхема (модифицировать самостоятельно)

- Алгоритма средней точки

- построение эллипса с помощью библиотечной функции

Пользователь выбирает из списка определенный алгоритм, задает координаты центра, полуоси, цвет рисования.

Визуальные характеристики исследуются путем рисования того же эллипса цветом фона, но с помощью другого алгоритма.

П 1 и 2 предусматривают рисование одиночных кривых.

3. Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических окружностей.

Пользователь выбирает из списка определенный алгоритм, задает координаты центра, цвет рисования, три из следующих четырех параметров: начальный радиус, конечный радиус, шаг изменения радиуса, количество окружностей.

Визуальные характеристики исследуются путем рисования того же спектра окружностей цветом фона, но с помощью другого алгоритма.

4. Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических эллипсов.

Пользователь выбирает из списка определенный алгоритм, задает координаты центра, цвет рисования, начальные значения полуосей, шаг изменения одной из полуосей, количество эллипсов.

Визуальные характеристики исследуются путем рисования того же спектра эллипсов цветом фона, но с помощью другого алгоритма.

**Дополнительное задание.**

Сравнить временные характеристики разных алгоритмов, построив в одном поле вывода (в одной системе координат и одном масштабе) графики зависимости времени работы алгоритма от радиуса (для окружности).

Для эллипсов построить аналогичную зависимость (зависимость времени работы алгоритма от изменения полуоси. Имеется в виду, что вторая полуось тоже будет изменяться см. п.4).

**Рассмотрим реализованные алгоритмы.**

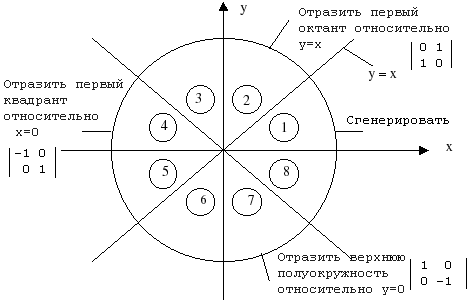
1) Использование канонического уравнения

**Алгоритм построения окружности**

Идея данного алгоритма проста: имея каноническое уравнение окружности можно выразить одну из координат (то есть получить *у = …)*, и далее, устанавливая приращение аргумента = 1, находить в каждой точке значение функции. Пусть для определенности мы выражаем *у.* Тогда, учитывая, что у должен иметь меньшее приращение (чтобы шаг по х, равный единице, не приводил к прерываниям функции), нам следует чертить только ту часть окружности, которая имеет наклон <= 45o (по модулю).

С данной проблемой можно легко справиться, потому что у окружности есть одно большое **преимущество** **(которое мы везде будем использовать)**:

Нам достаточно найти только 1/8 часть окружности, а всю окружность целиком можно получить путем 3 отражений: вокруг биссектрисы 1 четверти, вокруг оси Х и вокруг оси У (данные отражения указаны для окружности, лежащей в начале координат; если требуется отразить не в начале координат, то биссектриса и оси сдвинутся на величину сдвига центра фигуры). Ниже приведена картинка, как можно генерировать окружность.



В моем алгоритме я выражал *у*, а учитывая ограничения, накладываемые на приращение, я выбрал (2) кусок окружности (см. рисунок выше) для генерации, а все остальные получал путем трехкратного отражения.

Код программы

**def** canon\_c(x, y, r, colour):

points = list()

R = r \* r

**for** a **in** range(x, round(x + r / sqrt(2)) + 1):

b = y + sqrt(R - (a - x) \* (a - x))

points.append(Point(a, b, colour))

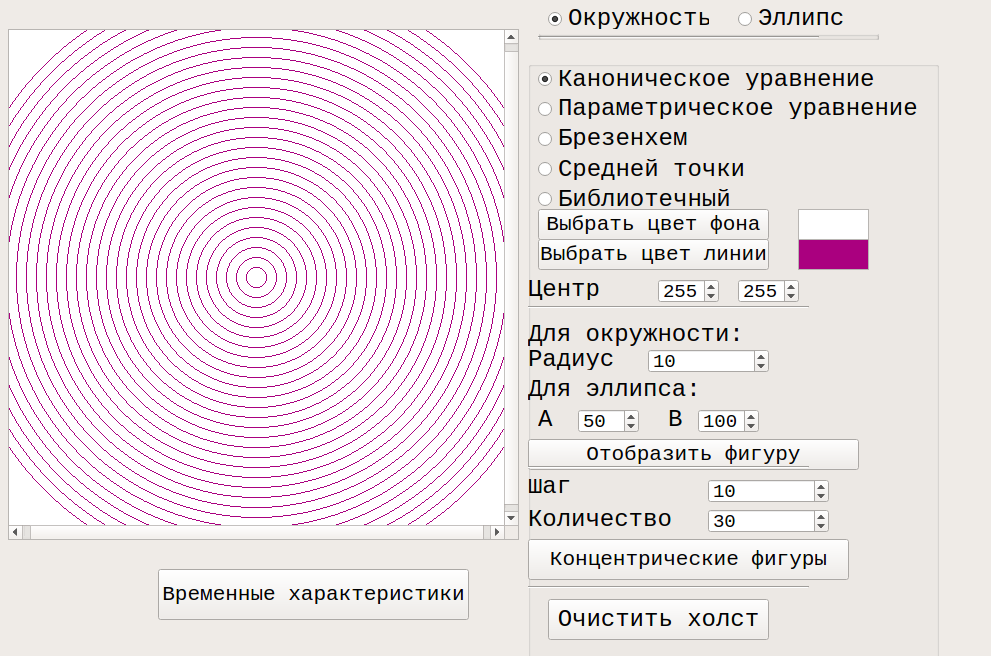
dup\_biss(points, x, y)

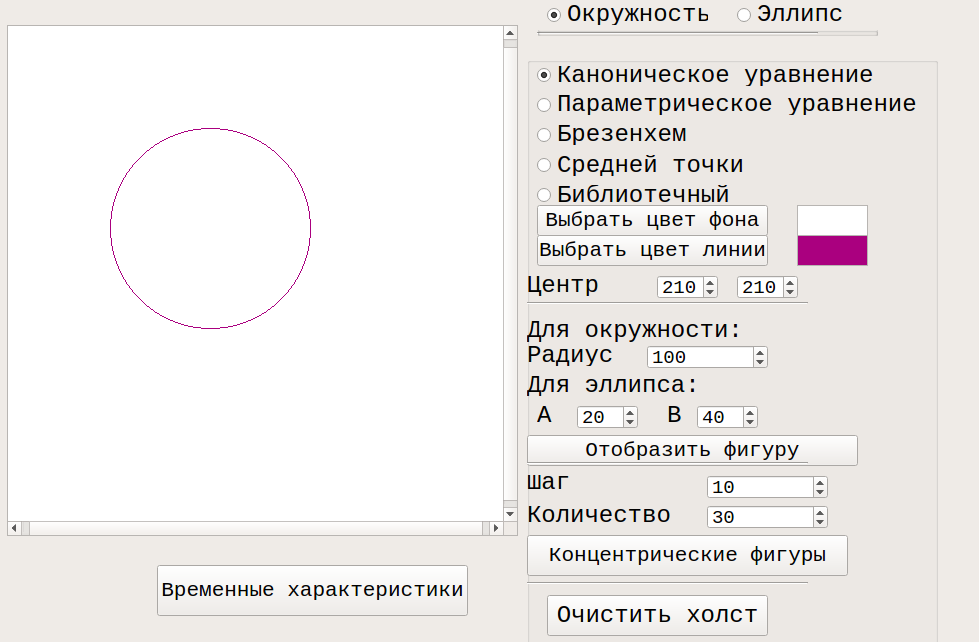
dup\_vert(points, x, y)

dup\_horiz(points, x, y)

**return** points

Концентрические окружности



Одна окружность

Алгоритм построения эллипса на основе канонического уравнения похож на аналогичный алгоритм для окружности, однако есть **важные** различия:

1. При построении эллипса требуется строить ¼, а не 1/8, так как эллипс не имеет симметрии относительно биссектрисы 1 четверти (в отличие от окружности).

2. В связи с надобностью строить ¼, появляется необходимость находить точку, в которой наклон касательной к эллипсу становится меньше 45о. Мы вынуждены найти эту точку, так как в ней приращения переменных х и у «меняются местами»: та переменная, которая имела большее приращение, начинает расти меньше, а другая — получает большее приращение. Соответственно в данной точке требуется другой переменной задать шаг = 1.

3. Каноническое уравнение эллипса содержит знаменатели под х и у, поэтому при преобразовании потребуется дополнительной домножить на знаменатель переменной, которую мы хотим выразить.

Мой код:

**def** canon\_ellips(xc, yc, a, b, colour):

points = list()

sqr\_a = a \* a

sqr\_b = b \* b

sqr\_ab = sqr\_a \* sqr\_b

lim1 = round(xc + a / sqrt(1 + sqr\_b / sqr\_a))

**for** x **in** range(xc, lim1):

y = yc + sqrt(sqr\_ab - (x - xc) \* (x - xc) \* sqr\_b) / a

points.append(Point(x, y, colour))

lim2 = round(yc + b / sqrt(1 + sqr\_a / sqr\_b))

**for** y **in** range(limit2, yc - 1, -1):

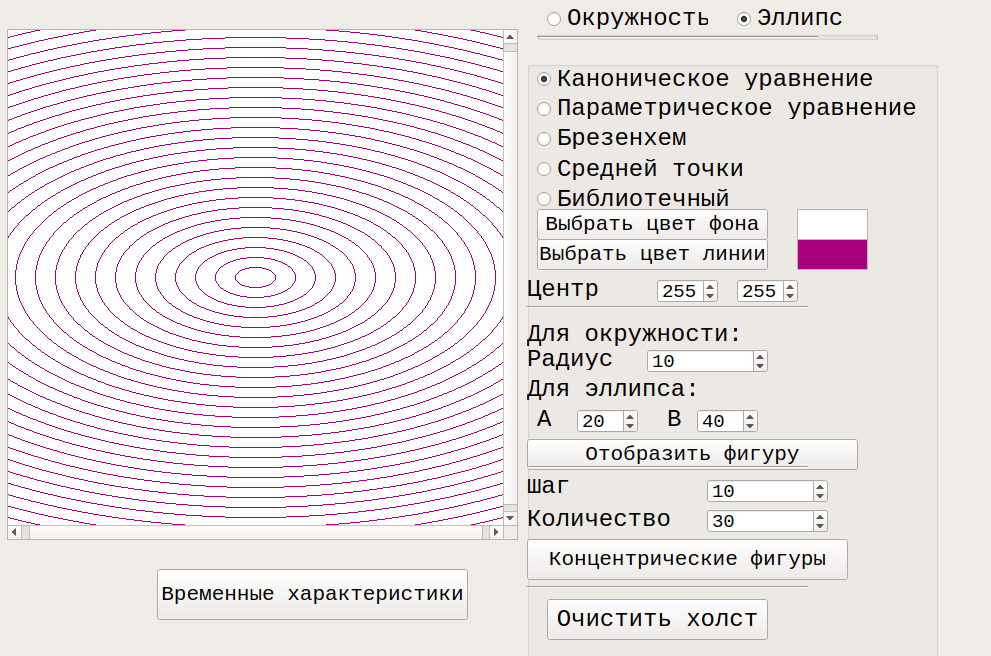
x = xc + sqrt(sqr\_ab - (y - yc) \* (y - yc) \* sqr\_a) / b

points.append(Point(x, y, colour))

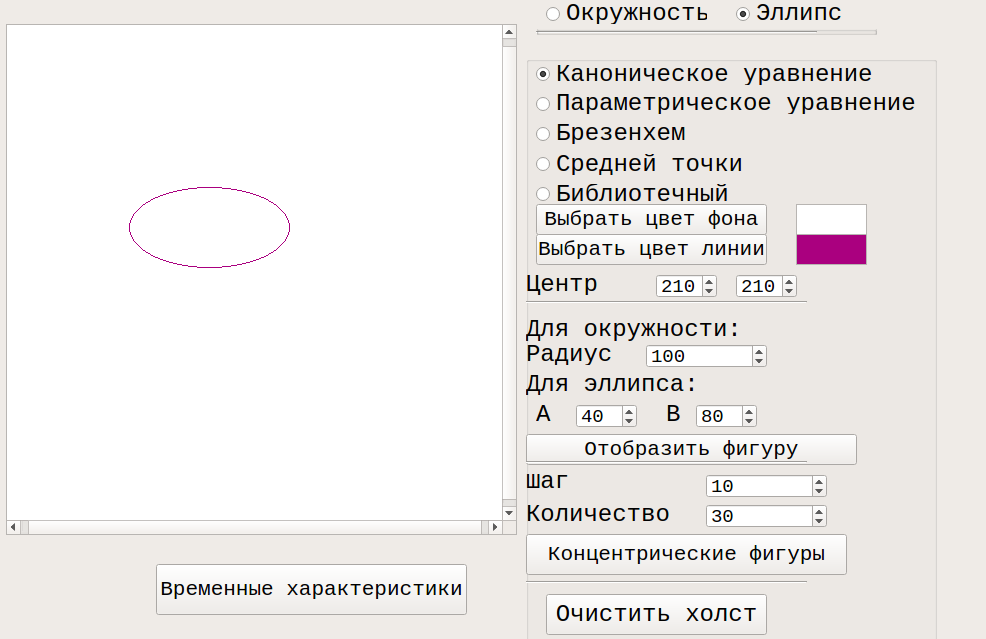
dup\_vert(points, xc, yc)

dup\_horiz(points, xc, yc)

**return** points

Концентрические эллипсы 

Один эллипс:



**Минусы алгоритма:** Долго работает

**Плюсы алгоритма:** Лёгкая реализация

2) Использование параметрического уравнения

**Построение окружности**

Идея данного алгоритма в следующем: мы выбираем некоторый шаг, равный 1/R, для переменной t, и с каждым новым вычислением увеличиваем t на эту величину. Данная величина была выбрана неслучайно: по 1 замечательному пределу *x / sin(x) = 1* и получается, что расстояние между рисуемыми пикселями пропорционально углу между ними (с вершиной в центре окружности). Так, при окружности радиусом 1, шаг угла будет = 1 радиан, расстояние между пикселями = 1 ширина пикселя; при увеличении радиуса в R раз, угол должен быть в R раза меньше, так расстояние между пикселями будет расти пропорционально этому радиусу (см. определения угла в 1 радиан).

В данном случае я снова чертил 1/8 окружности и далее отражал, для экономии времени (то есть получается угол будет меняться от 0 до pi / 8 с шагом 1 / R):

**def** param\_c(x, y, r, colour):

points = list()

step = 1 / r

**for** t **in** np.arange(0, pi / 8 + step, step):

a = x + r \* cos(t)

b = y + r \* sin(t)

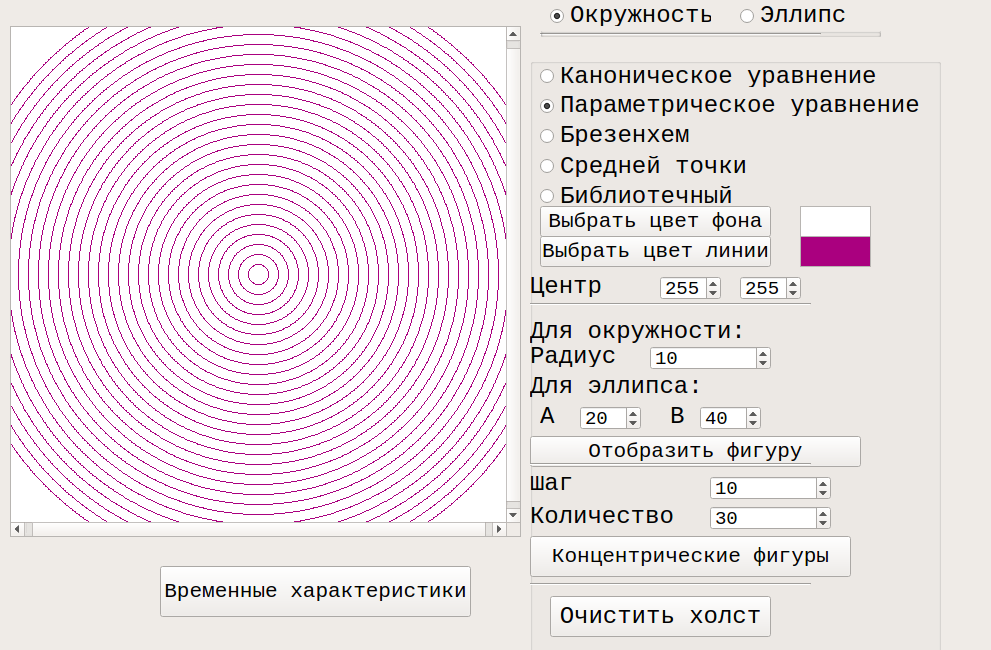
points.append(Point(a, b, colour))

dup\_biss(points, x, y)

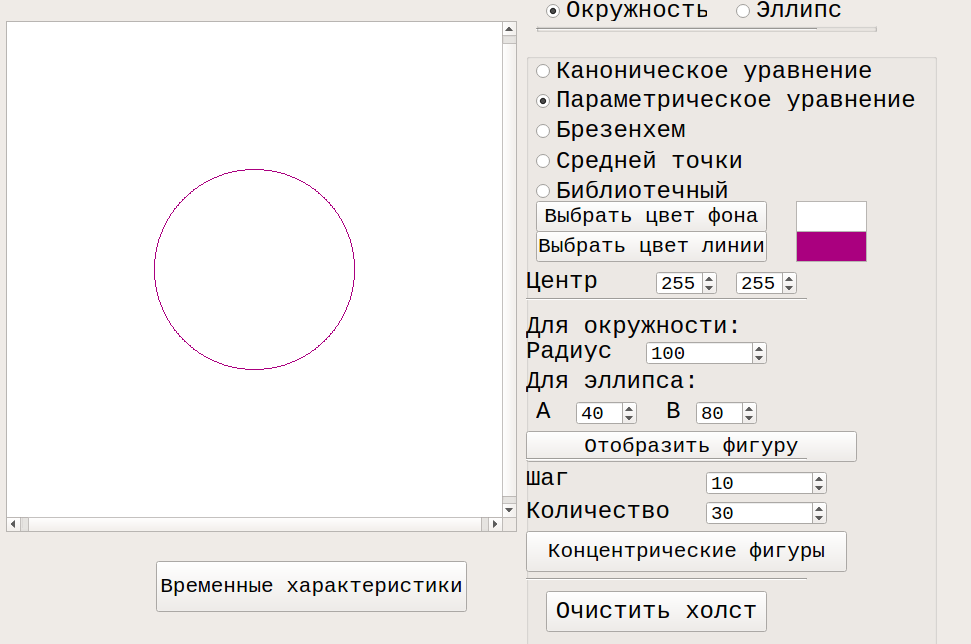
dup\_vert(points, x, y)

dup\_horiz(points, x, y)

**return** point

Концентрические окружности

Одна окружность



**Построение эллипса**

Параметрическое уравнение эллипса X=Xc+аcost, Y=Yc+ bsint, где Xc, Yc - координаты центра, a, b - полуоси, t - параметр (0<t <2).

Идея данного алгоритма такая же, как у алгоритма с окружностью, однако нам требуется (как и в алгоритме выше) строить ¼ часть фигуры вместо 1/8, а также появляется проблема с выбором шага, у которой есть 2 не самых лучших решения:

1. Выбрать больший радиус. В таком случае при большой разнице между полуосями мы построим много лишний точек.

2. Каждый раз высчитывать шаг исходя из текущего угла. Данное действие тоже весьма трудоемкое и при больших значениях полуосей эллипса (то есть при большом количестве точек) будет сильно потреблять вычислительные ресурсы.

Код программы:

**def** param\_ellips(x, y, r1, r2, colour):

points = list()

step = 1 / r1 **if** r1 > r2 **else** 1 / r2

**for** t **in** np.arange(0, pi / 4 + step, step):

a = x + r1 \* cos(t)

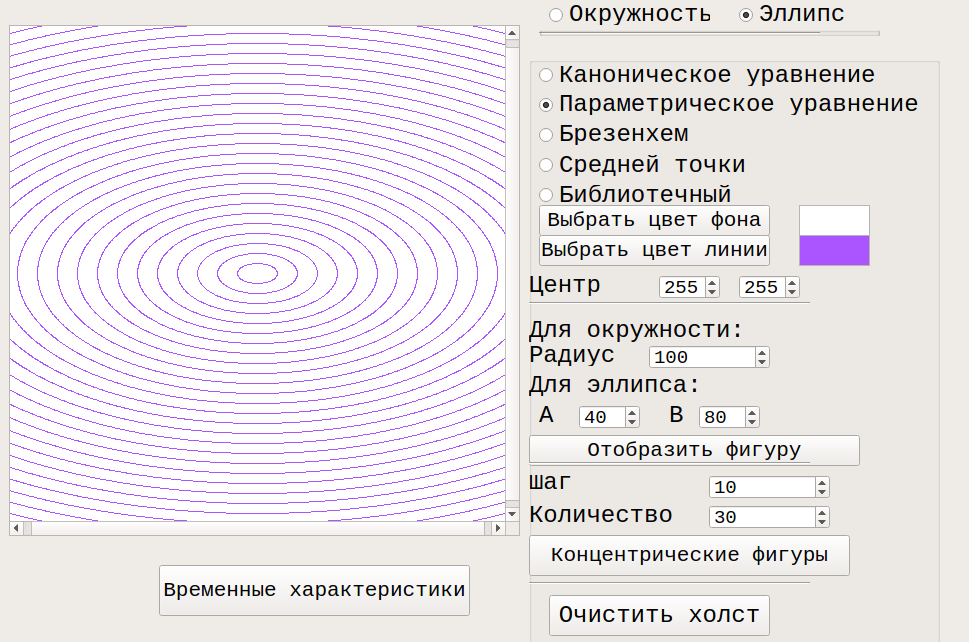
b = y + r2 \* sin(t)

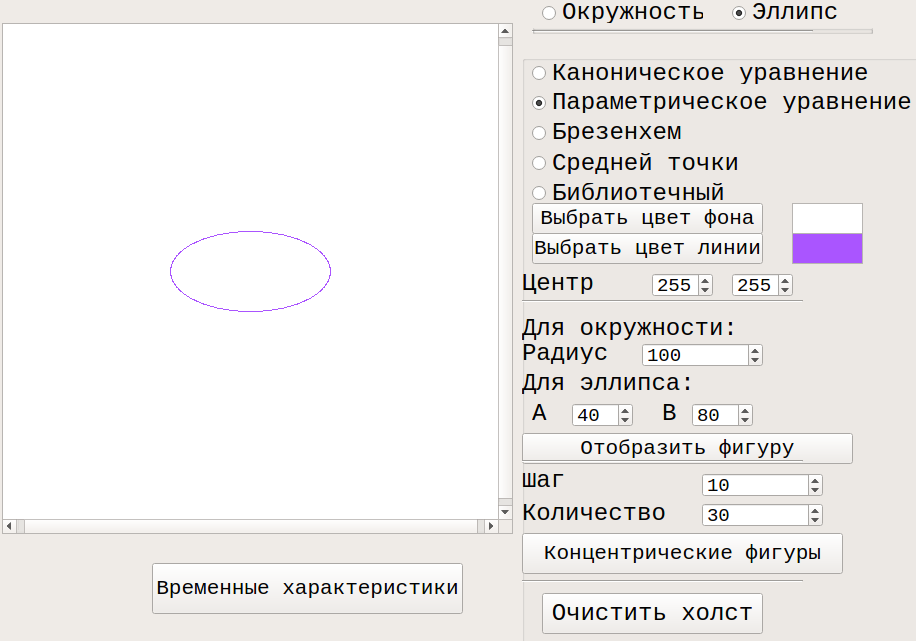
points.append(Point(a, b, colour))

dup\_vert(points, x, y)

dup\_horiz(points, x, y)

**return** points

Концентрические эллипсы:

Один эллипс

Плюсы:

Простая реализация

Минусы:

Долго работает

3) Алгоритм Брезенхема

**Построение окружности**

Рассмотрим построение части окружности, лежащей в первом квадранте. Предположим, что центр окружности лежит в начале координат, ее радиус равен R, ось абсцисс направлена вправо, ось ординат - вверх. Пусть алгоритм начинает работу в точке с координатами X=0, Y=R, тогда он заканчивает работу в точке X=R, Y=0.

При таком направлении построения окружности Y как функция аргумента X будет являться монотонно убывающей (Y= R - X).

Алгоритм Брезенхема для генерации окружности в первом квадранте может быть записан в следующем виде:

1.Ввод исходных данных R (радиус окружности) и при необходимости Xc, Yc (координаты центра окружности).

2.Задание начальных значений текущих координат пикселя X=0, Y=R, параметра D=2(1-R), установка конечного значения ординаты пикселя Yk=0.

3.Высвечивание текущего пикселя (X, Y).

4.Проверка окончания работы: если Y <Yk, то переход к п.11.

5.Анализ значения параметра D: если D <0, то переход к п.6; если D=0, то переход к п.9; если D>0, то переход к п.7.

6.Вычисление параметра D1=2D+2Y-1 и анализ полученного значения: если D1<0, то переход к п.8;

если D1>0, то переход к п.9.

7.Вычисление параметра D2=2D-2X-1 и анализ полученного значения: если D2<0, то переход к п.9;

если D2>0, то переход к п.10.

8.Вычисление новых значений X и D (горизонтальный шаг): X=X+1; D=D+2X+1. Переход к п.3.

9.Вычисление новых значений X,Y и D (диагональный шаг): X=X+1; Y=Y-1; D=D+2(X-Y+1). Переход к п.3.

10.Вычисление новых значений Y и (вертикальный шаг):

Y=Y-1; D=D-2Y+1. Переход к п.3.

11.Конец.

Код программы:

**def** brezenham\_c(xc, yc, r, colour):

points = list()

x = 0

y = r

points.append(Point(x + xc, y + yc, colour))

delta = 2 - r - r

**while** x < y:

**if** delta <= 0:

d1 = delta + delta + y + y - 1

x += 1

**if** d1 >= 0:

y -= 1

delta += 2 \* (x - y + 1)

**else**:

delta += x + x + 1

**else**:

d2 = 2 \* (delta - x) - 1

y -= 1

**if** d2 < 0:

x += 1

delta += 2 \* (x - y + 1)

**else**:

delta -= y + y - 1

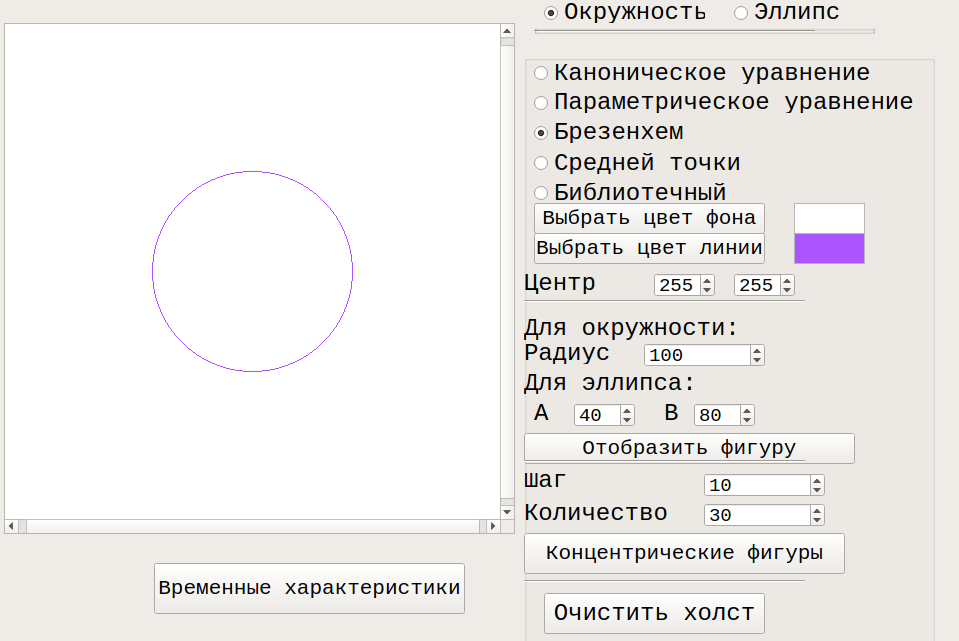
points.append(Point(x + xc, y + yc, colour))

dup\_biss(points, xc, yc)

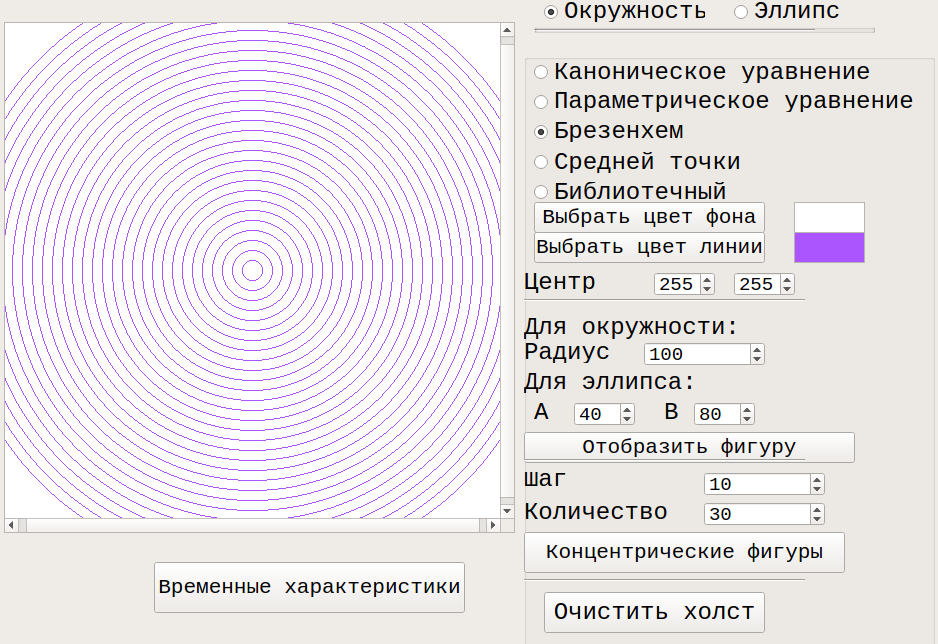
dup\_vert(points, xc, yc)

dup\_horiz(points, xc, yc)

**return** points

Одна окружность

Концентрические окружности



**Построение эллипса**

Модифицируем алгоритм построения окружности для построения эллипса.

Эллипс описывается каноническим уравнением X2/a2+Y2/b2=1 и для любого эллипса можно найти такую декартову систему координат, что это уравнение будет его описывать. При этом центр эллипса лежит в начале координат, а оси совпадают с координатными осями. При a=b данный алгоритм строит окружность, так как окружность действительно является частным случаем эллипса.

Как и в оригинальном алгоритме Брезенхема, выбор ближайшей точки основан на анализе знаков управляющих переменных, для вычисления которых используется исключительно целочисленная арифметика, что значительно повышает скорость выполнения алгоритма на любой ЭВМ.

Рассмотрим непосредственно алгоритм:

Для построения растровой развертки эллипса с осями, параллельными осям координат, и радиусами a, b воспользуемся каноническим уравнением X2/a2+Y2/b2=1, которое перепишем в виде f (x, y) ≡ b2x2+ a2y2– a2b2= 0.

За основу возьмем дугу, лежащую между точками (0, b) и (a, 0) в первом квадранте координатной плоскости.

В каждой точке (x, y) эллипса существует вектор нормали, задаваемый градиентом функции f. Дугу разобьём на две части: первая – с углом между нормалью и горизонтальной осью больше 45○ (тангенс больше 1) и вторая – с углом, меньшим 45○

Движение вдоль дуги будем осуществлять по часовой стрелке, начиная с точки (0, b).  
Направление нормали соответствует вектору grad (x, y) = (∂f/∂x, ∂f/∂y) = (2b2x, 2a2y).

Отсюда находим тангенс угла наклона вектора нормали: t = a2y/b2x. Приравнивая его единице, получаем что координаты точки деления дуги на вышеуказанные части удовлетворяют равенству b2x = a2y. Поэтому критерием того, что мы переходим ко второй области будет соотношение a2(y-1/2) ≤ b2(x+1) (координаты дополнительной точки (x+1, y-1/2)), или , переходя к целочисленным операциям, a2(2y-1) ≤ 2b2(x+1).

Вдоль всей дуги координата Y является монотонно убывающей, но в первой части дуги она убывает медленнее, чем растёт X, а во второй быстрее. Поэтому в первой части дуги будем сначала увеличивать значение X, а во второй сначала уменьшать значение Y.

**Код программы**

**def** brezenham\_ellips(xc, yc, a, b, colour):

points = list()

x = 0

y = b

sqr\_b = b \* b

sqr\_a = a \* a

points.append(Point(x + xc, y + yc, colour))

delta = sqr\_b - sqr\_a \* (2 \* b + 1)

**while** y > 0:

**if** delta <= 0:

d1 = 2 \* delta + sqr\_a \* (2 \* y - 1)

x += 1

delta += sqr\_b \* (2 \* x + 1)

**if** d1 >= 0:

y -= 1

delta += sqr\_a \* (-2 \* y + 1)

**else**:

d2 = 2 \* delta + sqr\_b \* (-2 \* x - 1)

y -= 1

delta += sqr\_a \* (-2 \* y + 1)

**if** d2 < 0:

x += 1

delta += sqr\_b \* (2 \* x + 1)

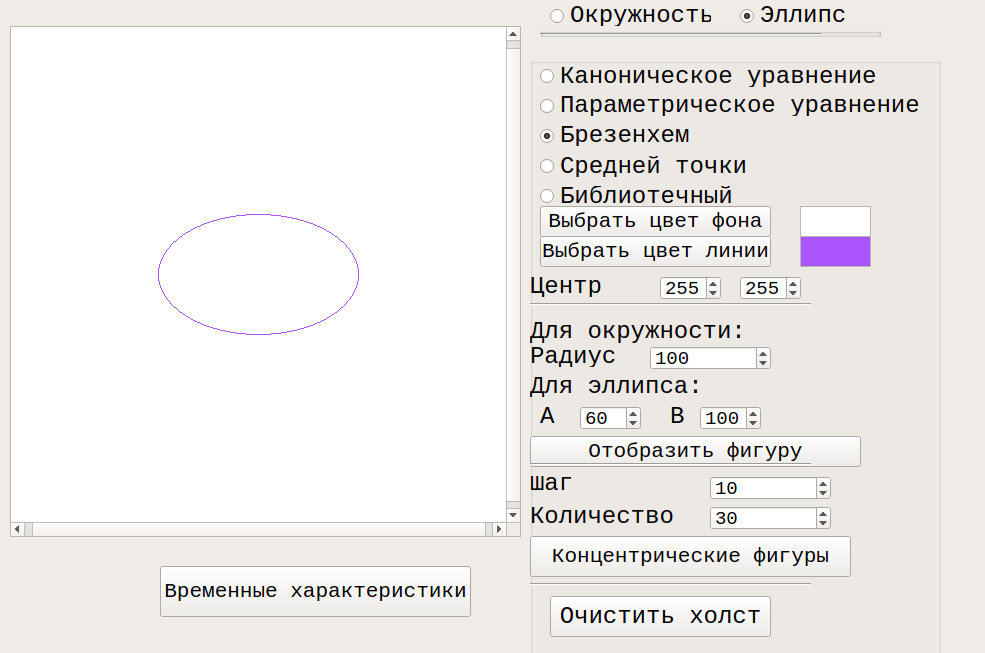
points.append(Point(x + xc, y + yc, colour))

dup\_vert(points, xc, yc)

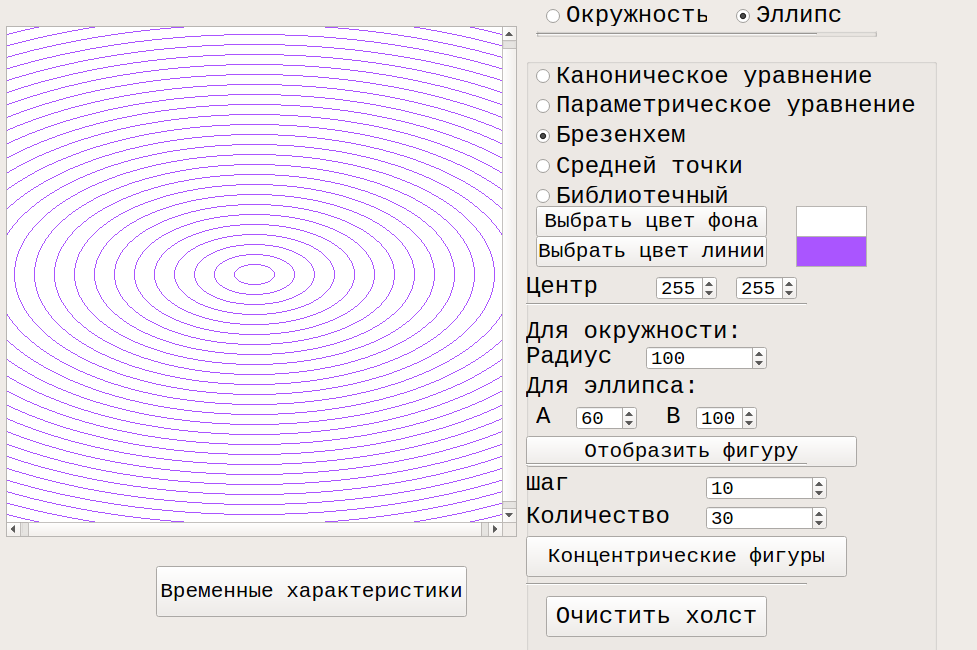
dup\_horiz(points, xc, yc)

**return** points

Один эллипс



Концентрические эллипсы



Плюсы алгоритма:

высокая скорость работы, минимальное потребление памяти;

значительная схожесть построенной фигуры с идеальной фигурой;

использование исключительно целочисленной арифметики.

Минусы:

Для реализации алгоритма необходимо рассмотреть множество случаев, что делает реализацию сложнее.

4) Алгоритм средней точки

**Построение окружности**

1.Введите радиус **r** и центр круга (xc, yc) и получите первую точку на окружности

2.Рассчитайте начальное значение параметра решения как

P0= 5/4 – r

3. На каждой позиции XK, начиная с K = 0, выполните проверку знака P.

**Если** p < 0: (*средняя точка внутри окружности, ближе верхний пиксел, горизонтальный шаг)*

p += 2 \* x + 1

**Иначе** (*средняя точка вне окружности, ближе диагональный пиксел, диагональный шаг)*

p += 2 \* x - 2 \* y + 5

y -= 1

4. Определите точки симметрии в других семи октантах.

5.Переместите каждую позицию расчетного пикселя (X, Y) на круговую траекторию с центром в (XC, YC) и нанесите значения координат.

6.Повторите шаги с 3 по 5, пока X> = Y.

Код программы:

**def** middle\_point\_c(xc, yc, r, colour):

points = list()

x = r

y = 0

points.append(Point(xc + x, yc + y, colour))

p = 1 - r

**while** x > y:

y += 1

**if** p >= 0:

x -= 1

p -= x + x

p += y + y + 1

points.append(Point(xc + x, yc + y, colour))

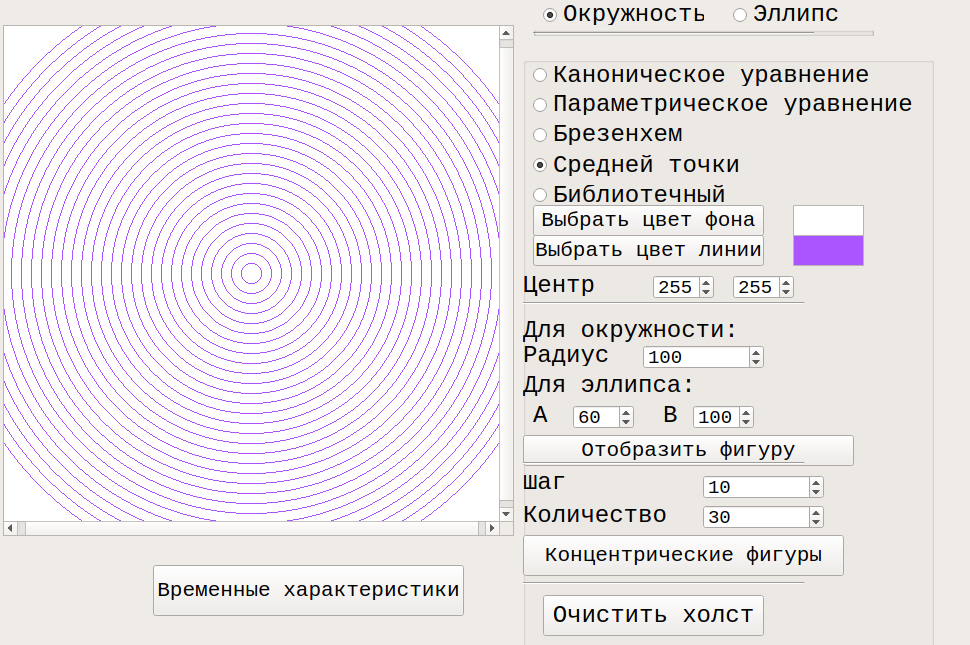
dup\_biss(points, xc, yc)

dup\_vert(points, xc, yc)

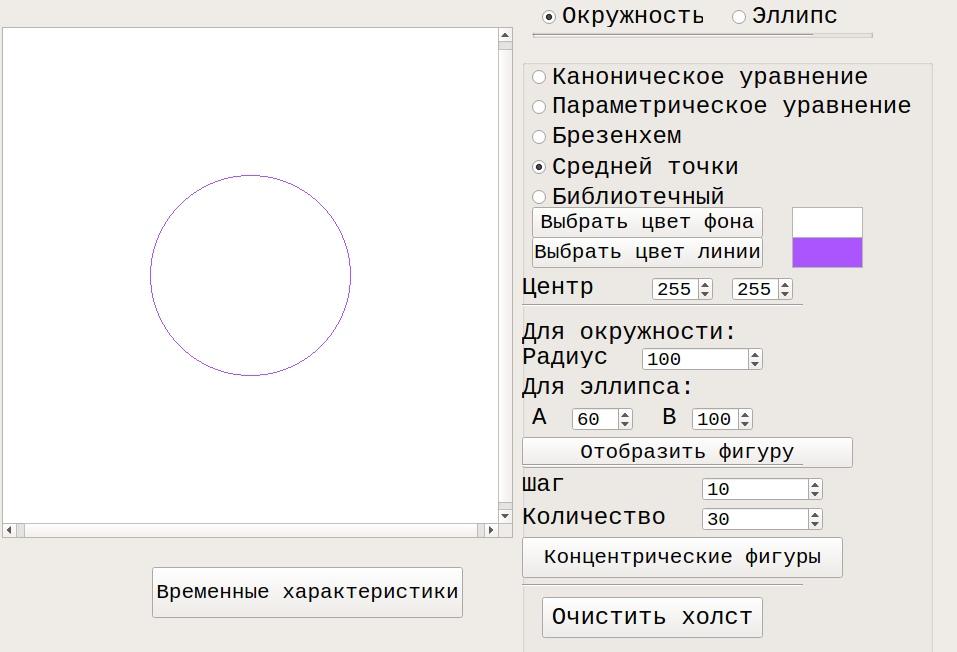
dup\_horiz(points, xc, yc)

**return** points

Концентрические окружности:



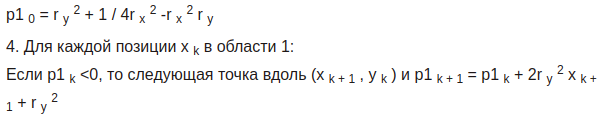
Одна окружность



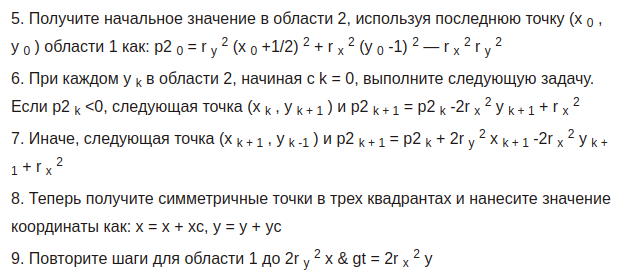
**Построение эллипса**

Возьмите входной радиус вдоль оси x и оси y и получите центр эллипса.

1. Первоначально мы предполагаем, что эллипс центрируется в начале координат, а первая точка имеет вид: (x, y0) = (0, ry).
2. Рассчитайте начальное значение параметра решения как: p10= ry^2+ 1 / 4 \* rx^2 - rx^2 \* ry
3. Для каждой позиции xk в области 1:







Код программы

**def** middle\_point\_ellips(xc, yc, a, b, colour):

points = list()

sqr\_a = a \* a

sqr\_b = b \* b

limit = round(a / sqrt(1 + sqr\_b / sqr\_a))

x = 0

y = b

points.append(Point(x + xc, y + yc, colour))

fu = sqr\_b - round(sqr\_a \* (b - 1 / 4))

**while** x < limit:

**if** fu > 0:

y -= 1

fu -= 2 \* sqr\_a \* y

x += 1

fu += sqr\_b \* (2 \* x + 1)

points.append(Point(x + xc, y + yc, colour))

limit = round(b / sqrt(1 + sqr\_a / sqr\_b))

y = 0

x = a

points.append(Point(x + xc, y + yc, colour))

fu = sqr\_a - round(sqr\_b \* (a - 1 / 4))

**while** y < limit:

**if** fu > 0:

x -= 1

fu -= 2 \* sqr\_b \* x

y += 1

fu += sqr\_a \* (2 \* y + 1)

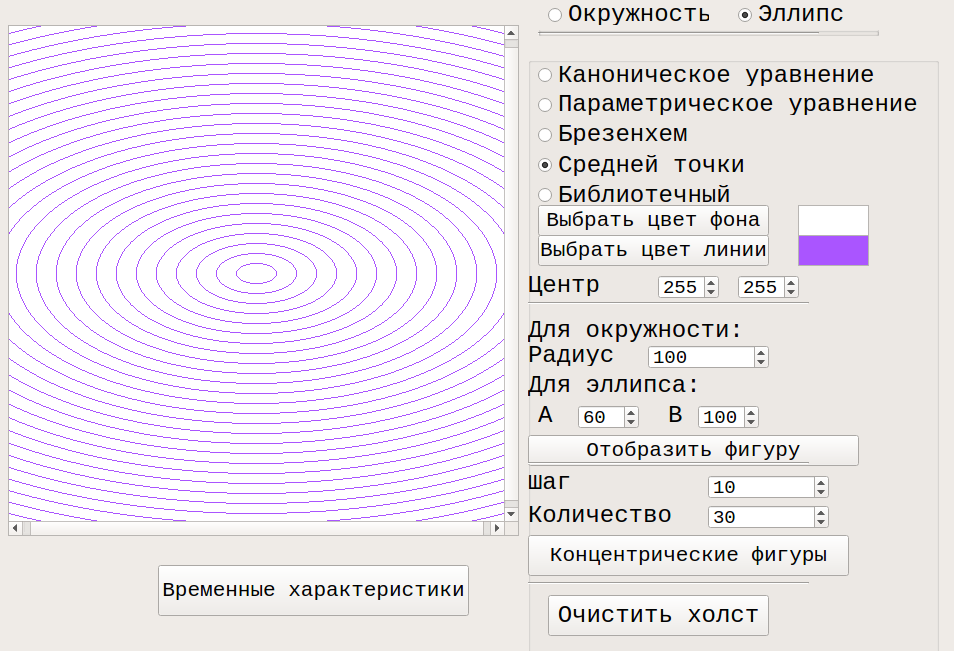
points.append(Point(x + xc, y + yc, colour))

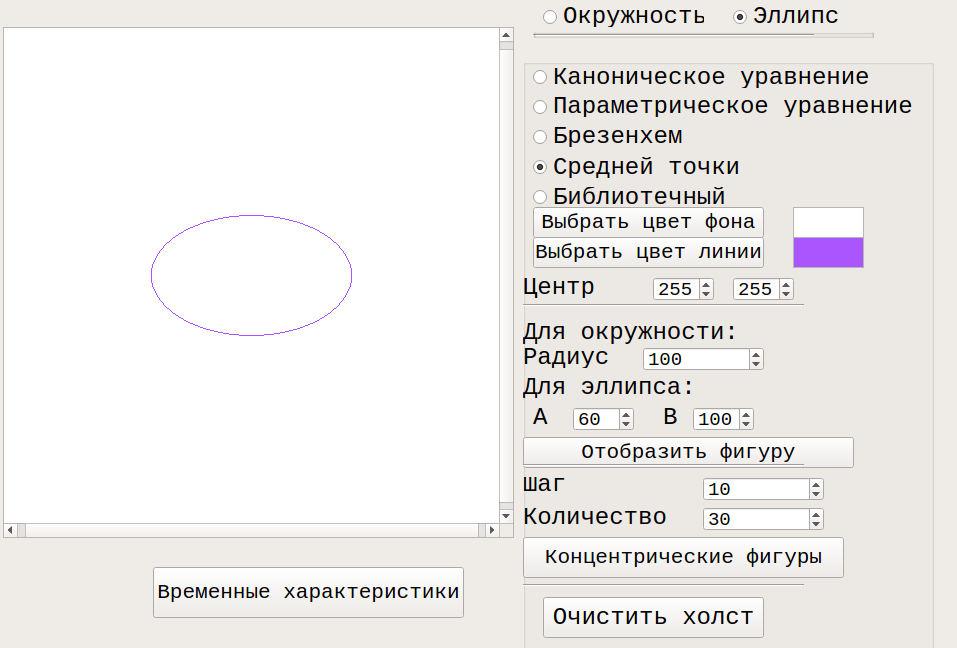
dup\_y(points, xc, yc)

dup\_x(points, xc, yc)

**return** points

Концентрические эллипсы

Один эллипс



Плюсы алгоритма:

высокая скорость работы, минимальное потребление памяти;

значительная схожесть построенной фигуры с идеальной фигурой;

Минусы:

Для реализации алгоритма необходимо рассмотреть множество случаев, что делает реализацию сложнее.

**Сравнение визуальных характеристик.**

**1. Окружность**

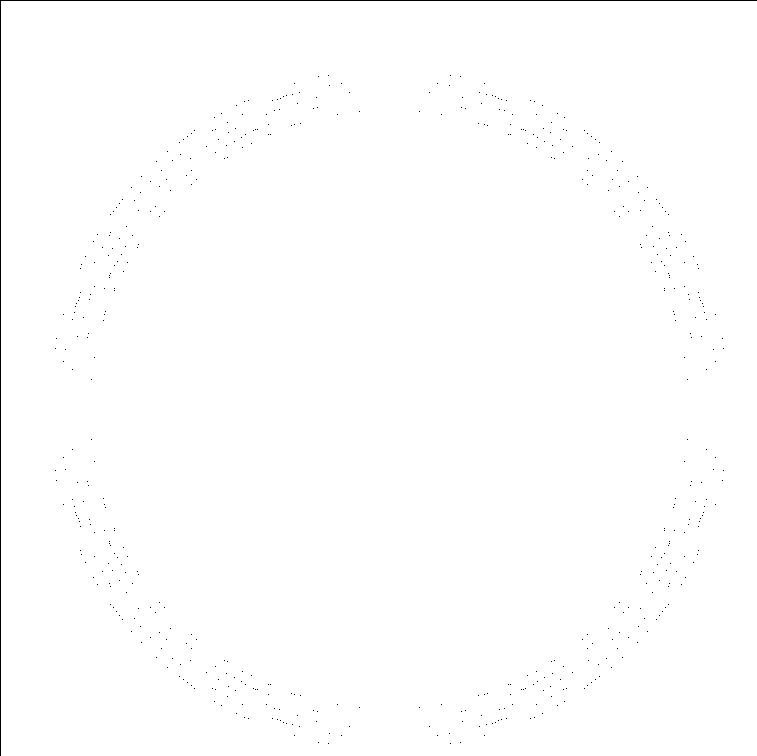
Из реализованных мною 4 алгоритмов 3 выдают абсолютно одинаковые окружности (библиотечная функция реализована одним из этих методов, так как дает такой же результат):

1) Каноническое уравнение.

2) Брезенхем.

3) Средняя точка.

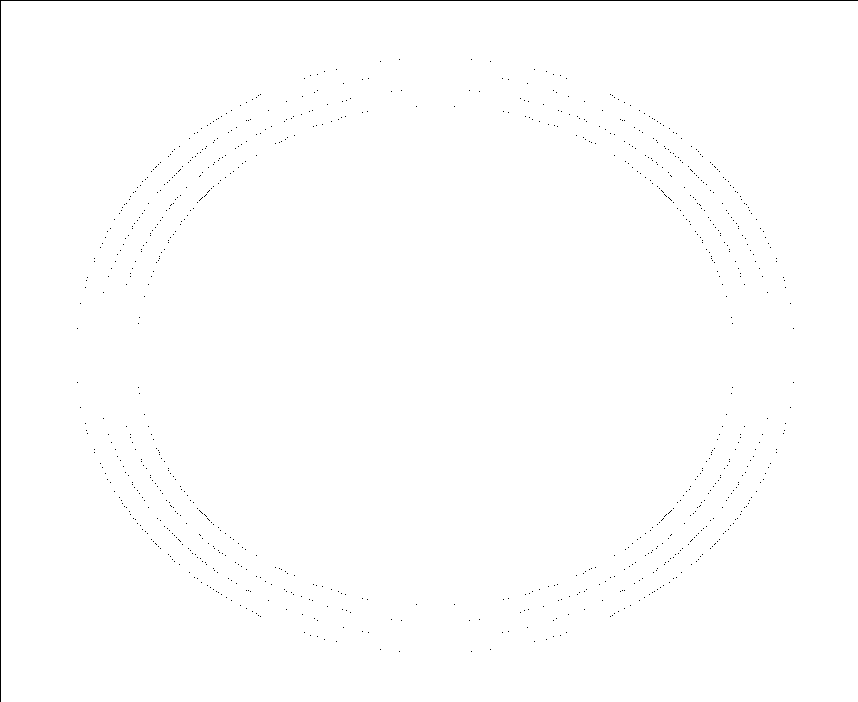
Параметрическое уравнение выдает другой результат (результат наложения параметрического уравнение на один из 3 алгоритмов выше):



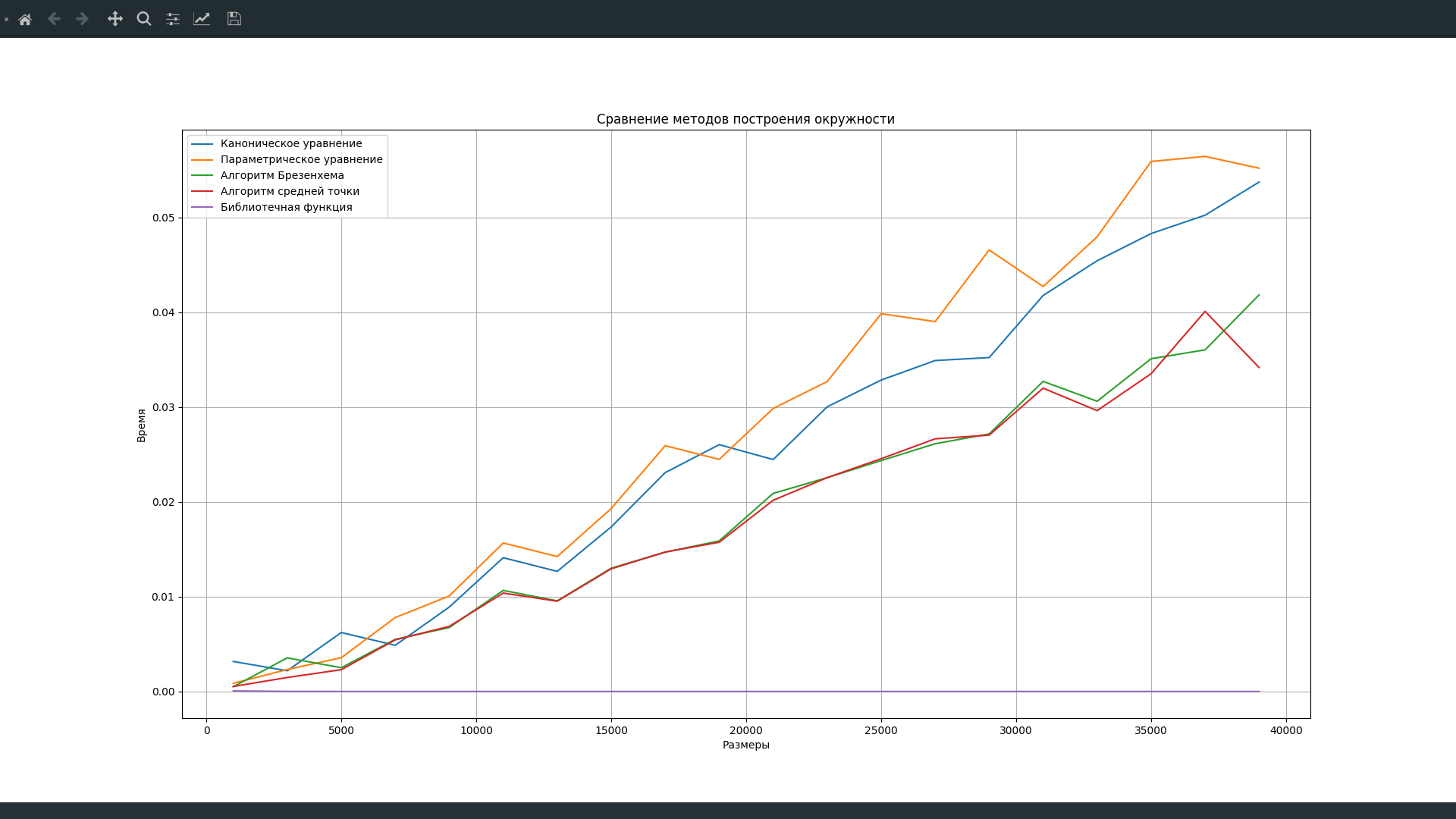
Данный результат связан с большой погрешностью метода, основанного на параметрическом уравнении: во-первых, шаг в нем измеряется в вещественных величинах, что уже дает некоторую погрешность (если шаг равен 1/3, то мы не можем получить точное значение (дробь несократима)), во-вторых, вычисление синуса и косинуса тоже не идеально (начиная с какого-то по счету символа не точно), что также вносит вклад в погрешность.

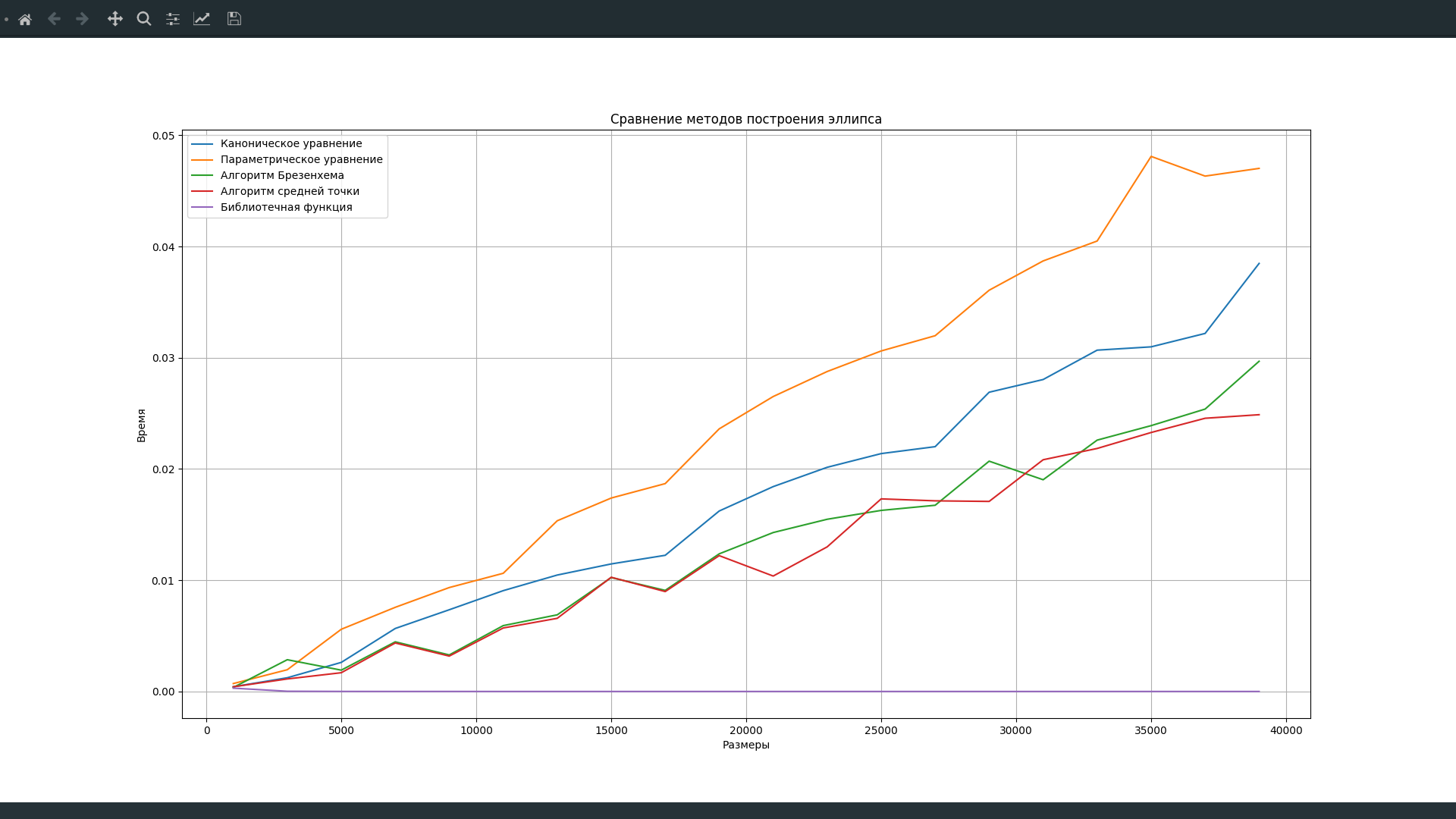
**2. Эллипс**

В эллипсах получена аналогичная картина: только параметрических метод отличается от остальных:



**Сравнение временных характеристик.**





Таким образом мы видим, что временные характеристики алгоритмов при построении окружностей и эллипсов похожи и хорошо видно следующее:

1. Алгоритмы, работающие с целыми числами, без квадратов, корней и синусов работают быстрее всего (метод средней точки работает чуть быстрее Брезенхема из-за того, что анализирует 2 пиксела вместо 3 и вследствие этого имеет меньше вычислений).

2. Алгоритм канонического уравнения и параметрических уравнений работают медленнее вследствие наличия вычислительно сложных операций (причем вычисление синуса и косинуса сложнее чем просто вычисление корня, поэтому алгоритм на основе системы параметрических уравнений работает медленнее алгоритма на основе канонического уравнения)

3. Так как в параметрическом методе эллипса я проводил какое-то кол-во лишних вычислений (связанных с рассмотрением большего радиуса), то этот метод для эллипсов еще больше отстает от остальных по времени, чем его аналог для окружностей.

**Общий вывод:**

При выборе алгоритма построения окружностей или эллипсов следует брать:

1. Алгоритм средней точки (самый быстрый и при этом точный)

2. Алгоритм Брезенхема (очень быстрый и точный)

3. Алгоритм на основе канонического уравнения (имеет вычислительные сложности, средний по скорости, но точный)

4. Алгоритм на основе параметрического уравнения (самый долгий в связи с вычислительными сложностями, при этом еще и имеет большую погрешность)